PHIL 7001: Fundamentals of AI, Data, and Algorithms

Week 10
Natural Language Processing

Maomei Wang
Email: mmmw@connect.hku.hk

Outline

Contents

- What is NLP
- How to do NLP
  - Thesaurus
  - Meaning and distribution
  - Word representation by counting
  - Word2Vec
  - NLMs
    - RNN
    - Seq2Seq
- State-of-the-art

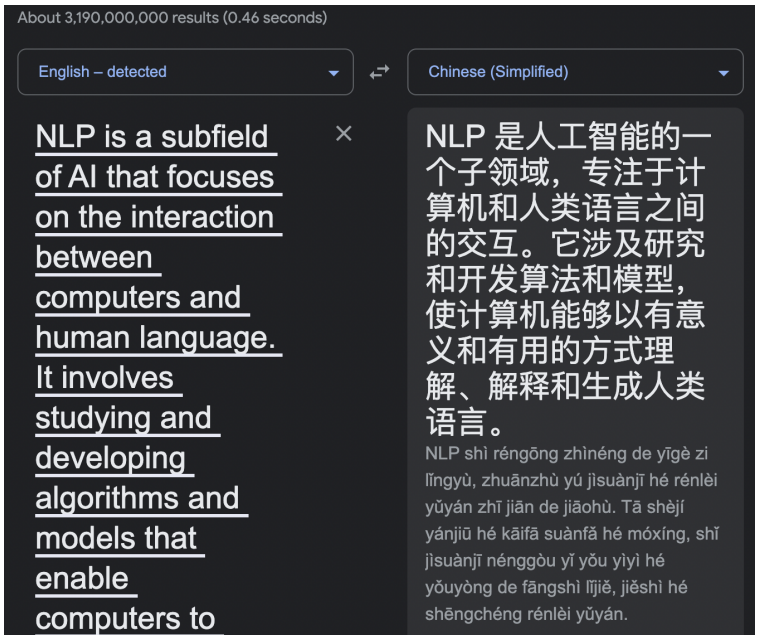
Introduction to NLP

NLP is a subfield of AI that focuses on the interaction between computers and human language.
It involves studying and developing algorithms and models that enable computers to understand human language.
In doing so, on the one hand, we have a better understanding of our language. On the other hand, the machines can communicate and even learn from us.

> What does it mean to "understand" human language? This is controversial. A currently hot debated topic is whether LLMs can understand human language — although we all know LLMs seem pretty good at delivering answers.

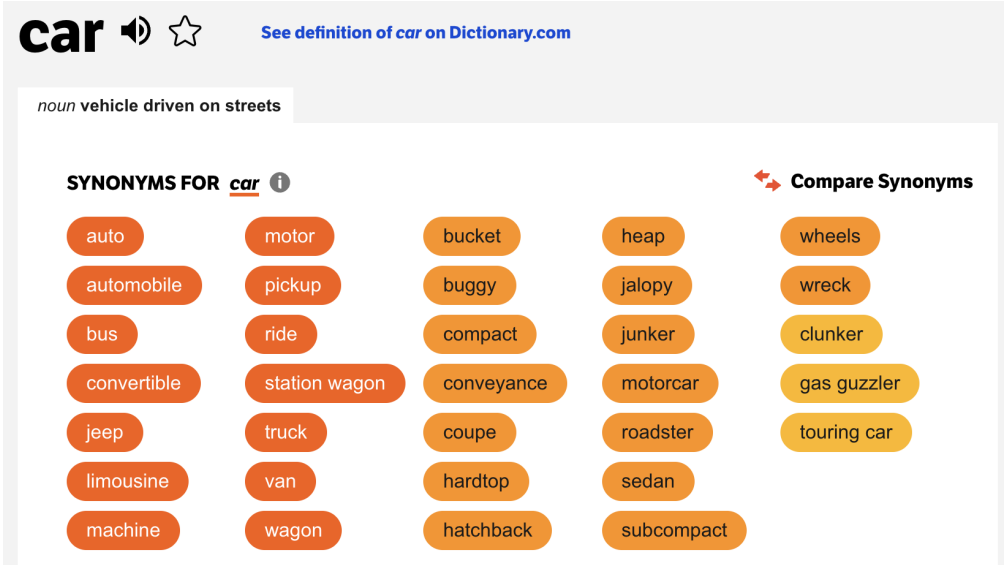
Applications of NLP

- Classifiers: classify set of documents into categories,
- Information Extraction: extract useful information from resumes
- Machine Translation: translate text from one human language into another
- Question Answering: find answers to natural language questions in a text collection or database
- etc



Based on synonyms

Provides synonyms (words with similar meanings) by humans.



We tell the machines these relationships among words and then let machines do things.

Example: WordNet

WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser.
Link: `https://wordnet.princeton.edu/`.
Problems:

- There are probably no examples of perfect synonymy: Even if many aspects of meaning are identical, they may differ based on politeness, slang, register, genre, etc
- How about new words?
- Changing meanings across times?
- Huge amounts of manpower...
- etc.

Similar approaches

- Antonymy
- Similarity
- Relatedness
- Connotation
- etc.

Similar problems!

Distributional hypothesis

> "The meaning of a word is its use in the language." — Ludwig Wittgenstein.

Distributional similarity vs. Meaning similarity

- The distributional hypothesis: semantically related or similar words will appear in similar contexts. (Zellig S. Harris (1954) Distributional Structure, WORD, 10:2-3, 146-162)
- "You shall know a word by the company it keeps." (Firth, J. R. (1957). Studies in Linguistic Analysis. Wiley-Blackwell.)

Word representation and Language model

Evidence shows that similar words are nearby in space.

- Word representation: each word's meaning is defined by its representation — a vector that captures its usage. This is also called *word embedding*.
- Language model: a representation that captures much of how words are used in a natural context will capture much of what we mean by meaning.

Paper: Li, J., Chen, X., Hovy, E., & Jurafsky, D. (2015). Visualizing and understanding neural models in NLP. *arXiv preprint arXiv:1506.01066.*

Introduction

Classical NLP gets the representation of words by counting.
They are trained on large corpora of text, such as books, articles, or web pages (Wikipedia, Google News, Shakespeare's books, etc.).
In what follows, let us use a single sentence to illustrate the mechanism.

> The "corpus": I want to know the distribution.

Co-occurrence matrix I

Here are all the words we have for this "corpus":

I	want	to	know	the	distribution	.

Then for "want", we know that (0,1 represents frequency)

	I	want	to	know	the	distribution	.
want	1	0	1	0	0	0	0

According to the distributional hypothesis, the meaning of "want" is determined by how "want" has been used.

Thus we can use the following vector to represent "want":

$$\text{want: } [1, 0, 1, 0, 0, 0, 0]$$

Illustration: "want" has appeared next to "I" and "to" once (window $= 1$).

Co-occurrence matrix II

Overall:

	I	want	to	know	the	distribution	.
I	0	1	0	0	0	0	0
want	1	0	1	0	0	0	0
know	0	0	1	0	1	0	0
the	0	0	0	1	0	1	0
distribution	0	0	0	0	1	0	1
.	0	0	0	0	0	1	0

The above is also called co-occurrence matrix.

I have omitted the one for the word "to"; please get this yourself.

Similarity measures

Let x, y denote two words. Let the function similarity(x, y) measure the word similarity between x and y. Below are example similarity measures:

- Cosine: let us abuse the notation and also use x, y to denote the vector that represents the two words, respectively,

$$\text{similarity}(x, y) = \frac{x \cdot y}{||x||||y||}$$

- Pointwise Mutual Information (PMI): such that the greater the number is, the more similar they are. It is defined as follows:

$$\text{similarity}(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

  Note here that if x and y are probabilistically independent, then $P(x, y) = P(x)P(y)$. The greater the number similarity(x, y), the more "dependent" these two words are.

Paper: Tan *et al.* (2002, July). Selecting the right interestingness measure for association patterns. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 32-41).

Problems

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Figure 6.6 Co-occurrence vectors for four words in the Wikipedia corpus, showing six of the dimensions (hand-picked for pedagogical purposes). The vector for *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

Source: Jurafsky, D. and Martin, J.H., Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognitio

- Vectors increase in size with vocabulary (the dimensionality of the vector, is often between 10,000 and 50,000 words): long, hard to train, require a lot of storage,
- Sparse (so many 0 appeared)

Word2Vec I

Word Representations in Vector Space (Word2Vec): *learn* word representation. (Check that you still remember representation learning vs. feature engineering!)
In what follows, let us continue with our simple corpus.

> The "corpus": I want to know the distribution.

Word2Vec II

Here are all the words we have for this "corpus":

I	want	to	know	the	distribution	.

Then for "want", we know that (0 represents no, 1 represents yes)

	I	want	to	know	the	distribution	.
want	0	1	0	0	0	0	0

Illustration: the word "want" is "want", but not others, so it gets 1 for "want", yet 0 for others. This is also called *one-hot* representation.

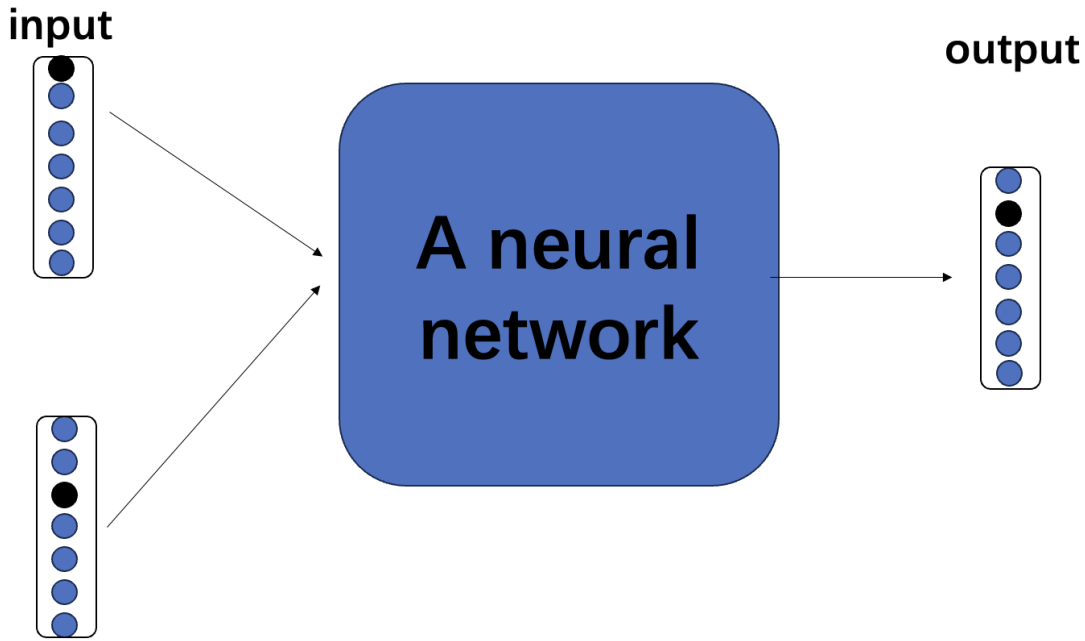
So we have a way to input these into a model to learn the word representations.

Word2Vec III: practice

Please fill out the table below:

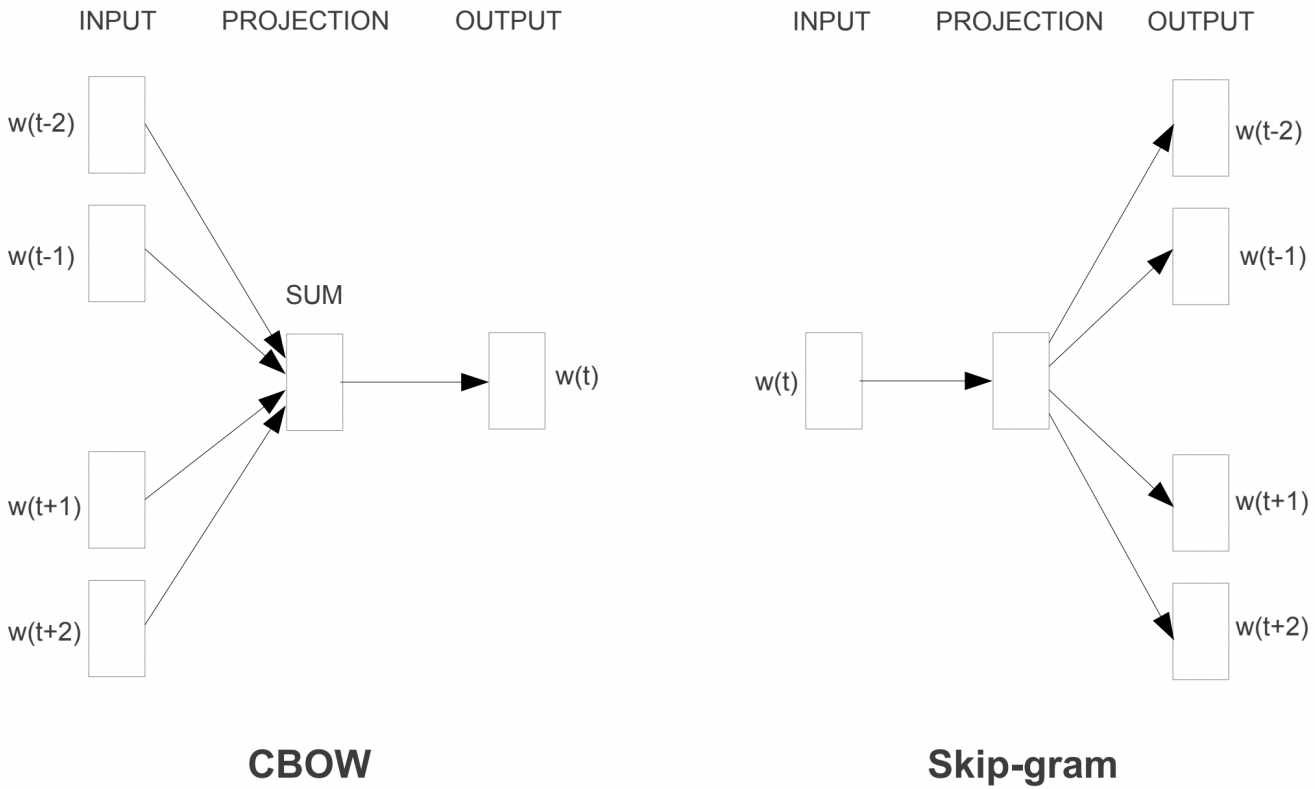
	I	want	to	know	the	distribution	.
I	1	0	0	0	0	0	0
want							
to							
know							
the							
distribution							
.							

Word2Vec IV: one example



Let the black circle represent 1, and the blue one represent 0, respectively. What's the input in the above mechanism? What's the output?

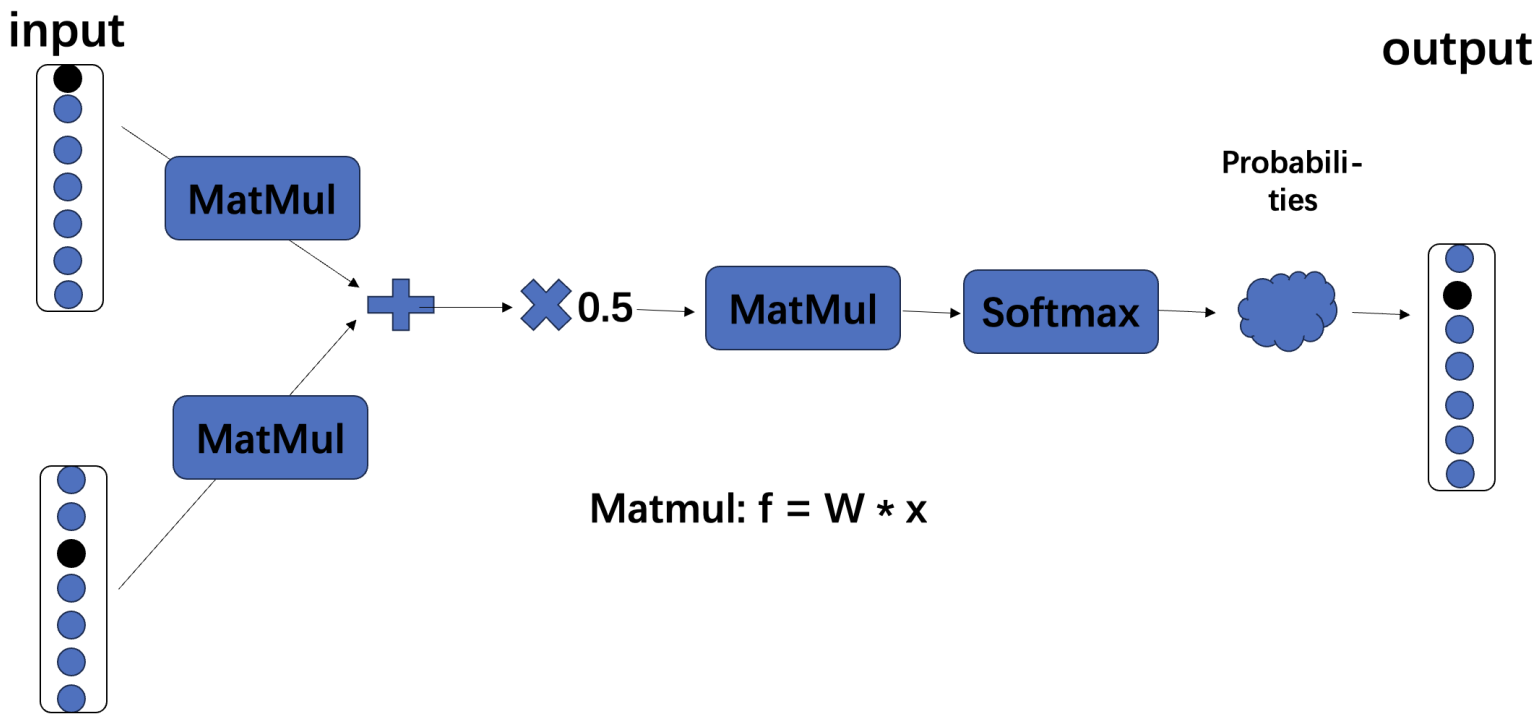
Word2Vec V: Classical two approaches



The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.
Paper: Mikolov, *et a.*. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

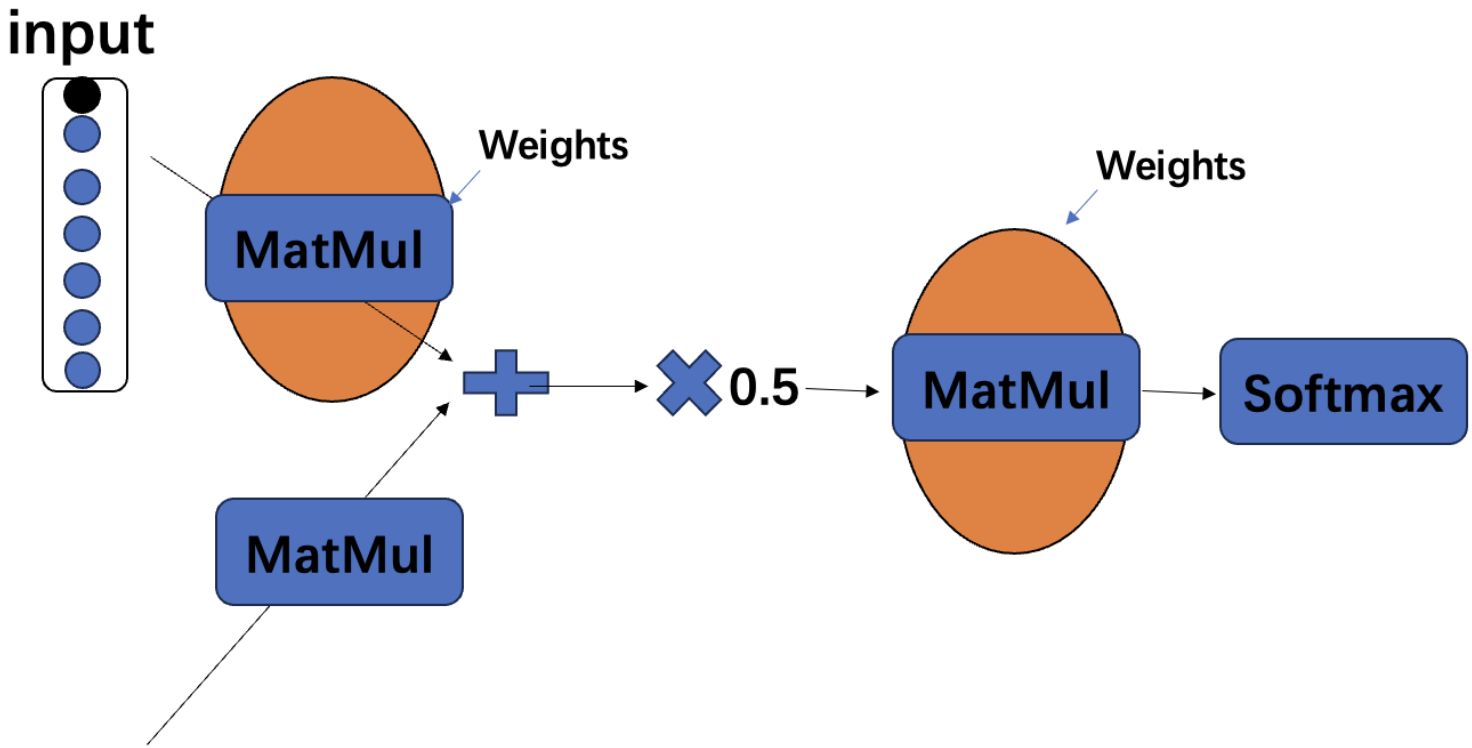
CBOW

Continuous bag-of-words model (CBOW):



This is how a CBOW model gives you an answer.

Weights as word representation



- Initiate weights and then learn to minimize loss.
- Using the learned weights vector as word representation.
- Short and dense word representation.

An example


```
from gensim.models import Word2Vec
sentences = [['I', 'want', 'to', 'know', 'the', 'distribution', 'ok'],
             ['the', 'distribution', 'is', 'not', 'clear'],
             ['yet', 'another', 'distribution'],
             ['is', 'even', 'worse'],
             ['and', 'the', 'final', 'distribution']]# Example sentences
model = Word2Vec(sentences, vector_size=10, window=2, min_count=1, workers=4) # Train the model
```

```
vector = model.wv['distribution']# Get the vector representation of a word
print(vector)
```

```
[-0.00535967  0.0023704   0.05103543  0.09009118 -0.09302995 -0.07116683
  0.06459596  0.089727   -0.05015733 -0.03763515]
```

```
similar_words = model.wv.most_similar('distribution', topn=3)# get other similar words
print(similar_words)
```

```
[('the', 0.5436095595359802), ('not', 0.43182066082954407), ('yet', 0.3793111741542816)]
```

Figure: Word2Vec of a simple corpus

Visualization

Similar words are nearby in space.

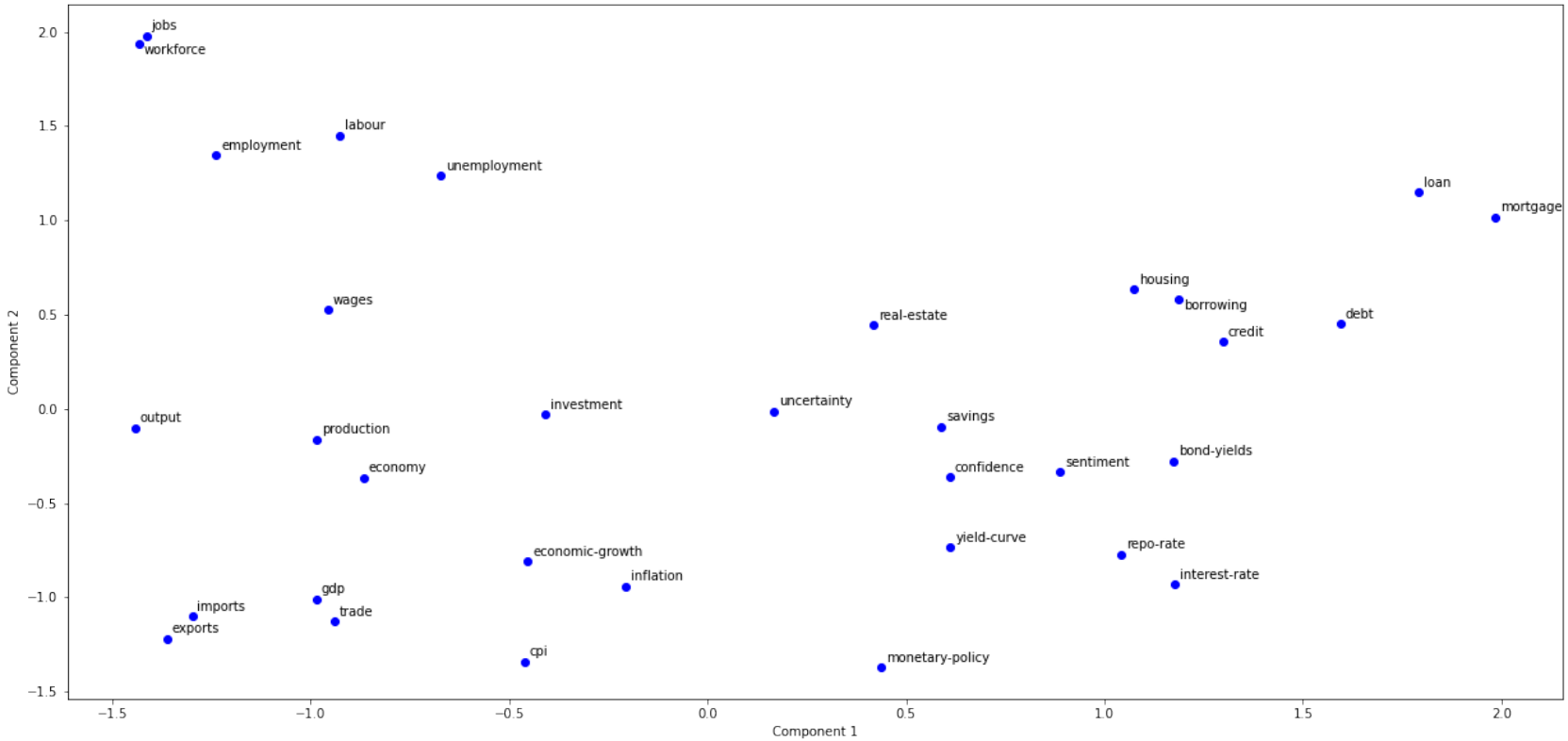


Figure: An example 2-dimension Word2Vec Visualization

Does this representation look good?

Language model as probability distribution

A language model is a probability function P over the set of word sequences. Formally, let $w_1, \ldots, w_n$ be a sequence of n words, $P(w_1, \ldots, w_n)$ is the probability of this sequence, and we know

$$P(w_1, \ldots, w_n) = \Pi_{t=1}^{n} P(w_t | w_1, \ldots, w_{t-1})$$

The machine outputs the next word based on probabilities. For example, when receiving "How are you?", the word (Fine), and then (.) (Thanks.), might get the highest probabilities, and the machine may thus output "Fine. Thanks.". Namely,

> Input: How are you?
> Machine: Fine. Thanks.

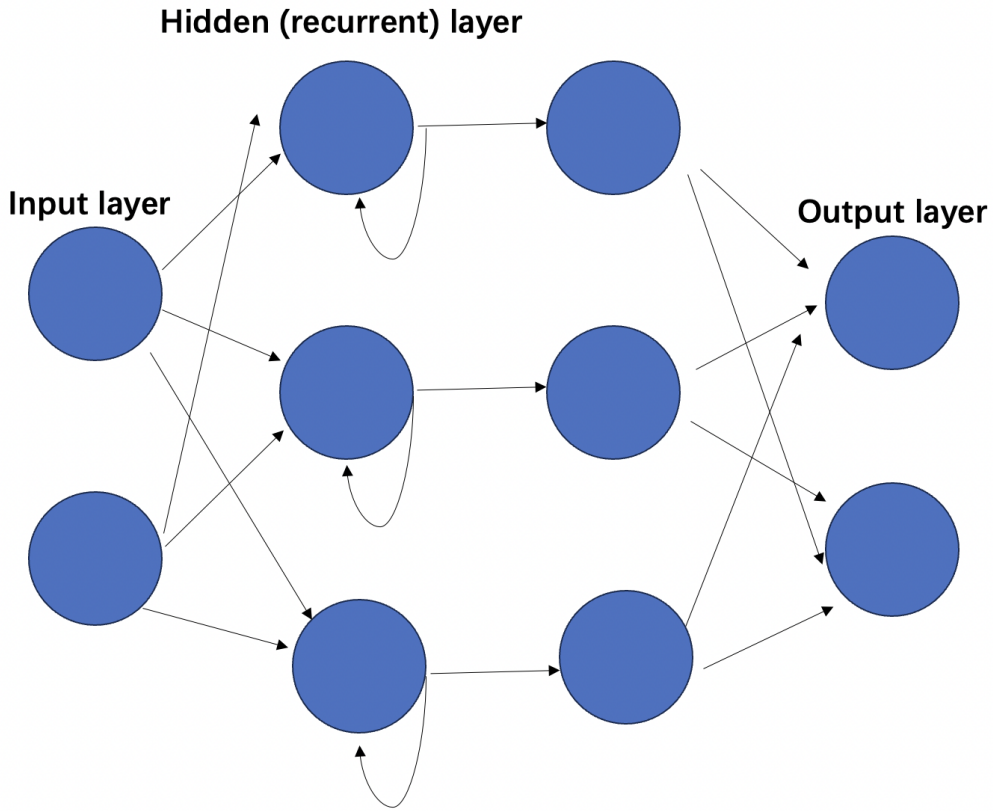
In doing so, the machine does not need to know the meanings of these words but only needs to know some (conditional) probabilities.
Most probabilistic language models approximate $P(w_t | w_1, \ldots, w_{t-1})$ using a fixed context of size $n-1$, i.e. using $P(w_t | w_{t-n+1}, \ldots, w_{t-1})$, as in n-grams. (Recall the Markov assumption.)

General idea

Neural Language Models (NLMs) are used to approximate the probability function. (Check whether you understand this sentence! It's one of the essential points made in the previous week!!)
It learns and adjusts iteratively after knowing the accuracy of its predictions.
We train them based on variants of SGD and BP algorithms. (And check whether you understand this sentence! It's another essential point made in the previous week!!)
Different NLMs have been put forward in the literature, and we will introduce some.

RNN I: a kind of neural network that can "remember"

Unlike CBOW, which is a kind of FNN, Recurrent Neural Networks (RNNs) have loops, and these loops cause recurrent connection that allows them to maintain internal memory and process sequences of arbitrary length.

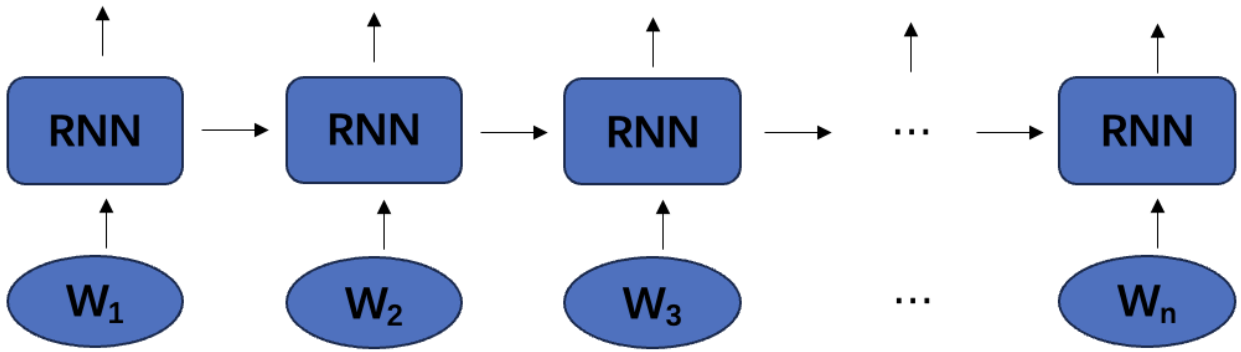


Memory is in need

On a sunny day, Jack was enjoying himself in the yard. Jack's friend, Mary, also joined him there and engaged in conversation.
Mary asked: "Do you think the weather is nice?"
Jack replied: "__".

What will Jack reply? How can machines do this?

RNN II

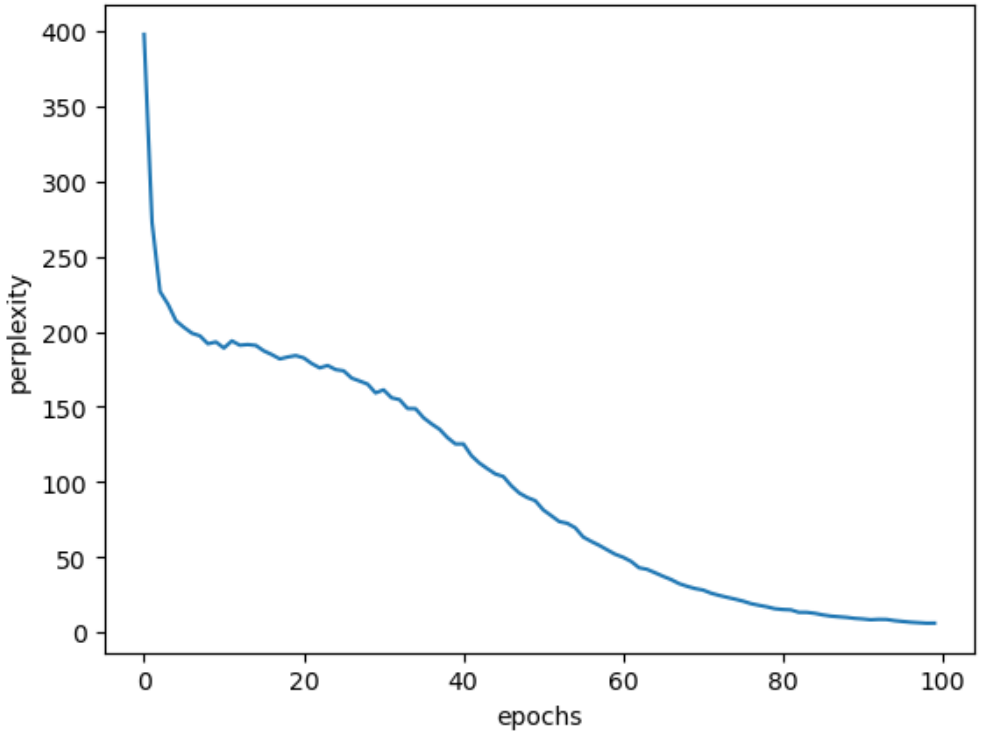


For example, let w_1 be "I", w_2 be "want", the followers be "to", "know" "the", and w_n be "distribution. When you at time t_n input w_n "distribution", the machine still receives information from "I" "want" "to" "know" "the", and this makes it like this time you are typing "I want to know the distribution". That's how the RNN "remembers". Also, note that the order of the input has been captured.

- Vanishing and Exploding Gradients: gradients become too large or too small, so the network can no longer learn (Check that you know learning DNNs relies on gradients! Although you are not required to know the details.).

Evaluation of RNNs

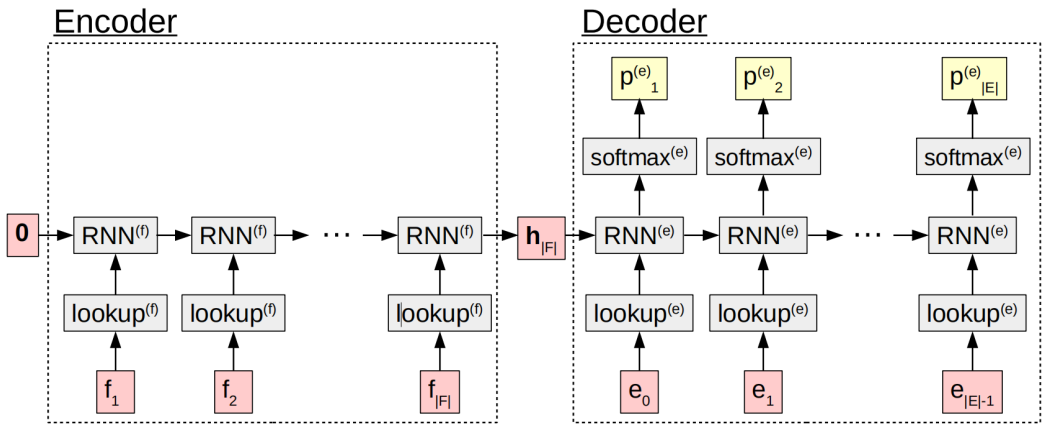
We commonly evaluate language models by its perplexity.



The number assigned are supposed to answwr "how confused is the model about its decision?" A lower perplexity indicates that the language model is more confident and accurate in predicting the next word in a sequence, thus considered better.

Seq2Seq

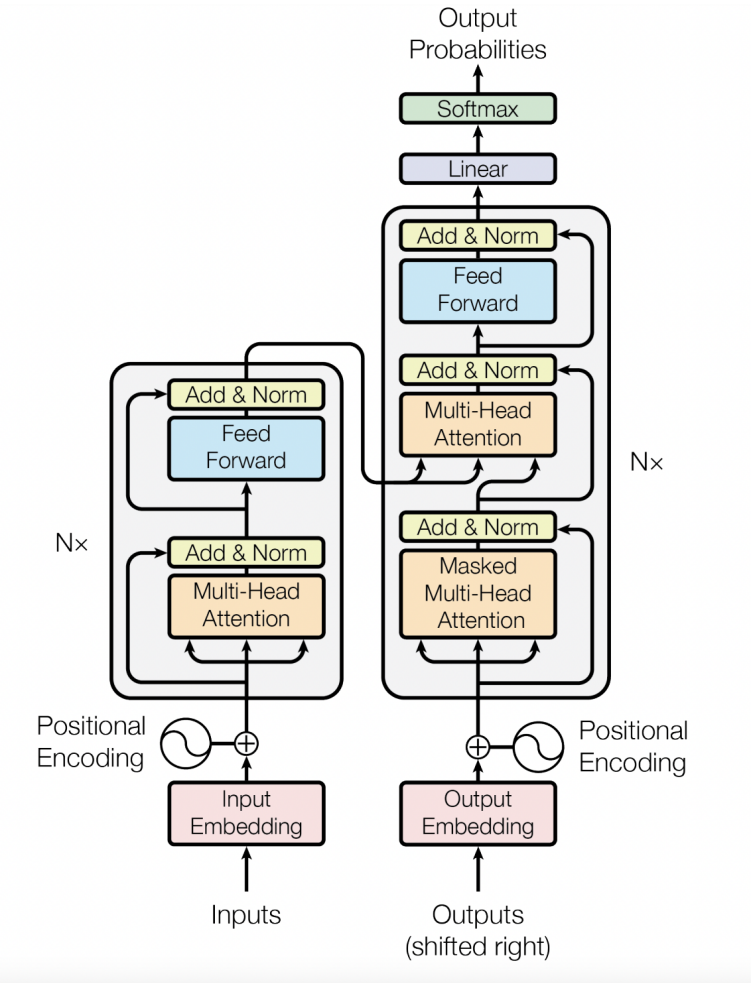
Sequence-to-sequence (Seq2Seq). It consists of two main components: an encoder and a decoder.



- The encoder processes the input sequence and converts it into a fixed-length representation called a *context vector*.
- The decoder then takes this context vector as input and generates the output sequence step by step.

Transformer

Problems with previous models: Hard to train and vanishing gradient problem.
A new model: Transformer.



Paper: Vaswani, A. *et al.* (2017). Attention is all you need. *Advances in neural information processing systems, 30.*

Pre-trained and fine-tuning

Current paradigm: Pre-trained and then fine-tuning.

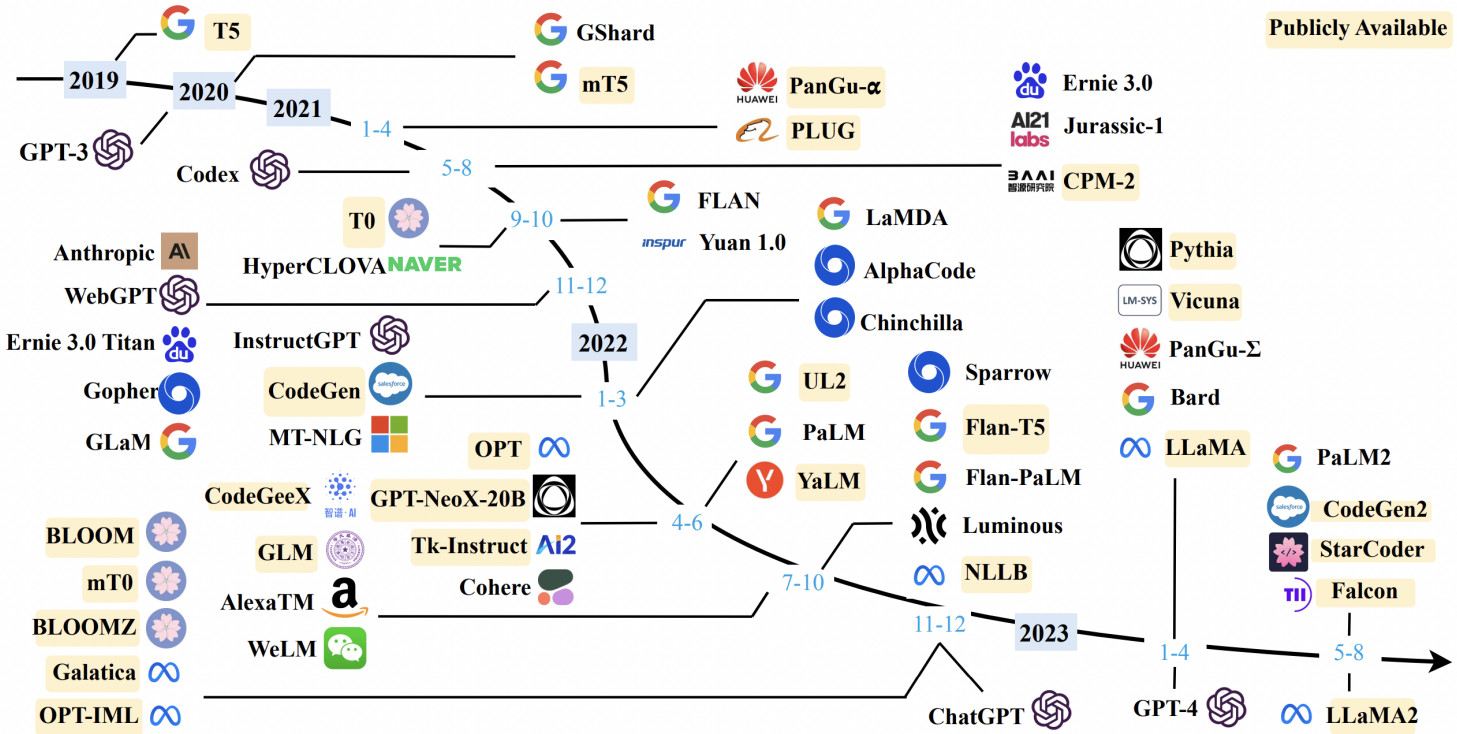
- Pre-trained: Pre-training a language model on large-scale unlabeled corpora. These models are called Pre-trained Language Models (PLMs).
- Fine-tuning: fine-tune the trained PLM on specific tasks.

Paper: Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Towards Large: LLMs

Researchers find that scaling PLM (e.g., scaling model size or data size) often leads to an improved model capacity on downstream tasks. Thus, some develop large-sized PLMs, and coin the term "Large Language Models (LLMs)".

Below are examples of large models with over 10B parameters.



Source: Zhao, *et al.* (2023). A survey of large language models. *arXiv preprint arXiv:2303.18223.*

Main points: NLP

1. NLP aims to enable machines to understand human language.
2. NLP has wild applications.
3. Classically, there are different ways to approach an understanding of language, such as approaches based on synonyms, antonyms, similarity, etc.
4. But these days the distributional hypothesis that builds a connection between meaning and distribution has been popular. Based on this idea, we build language models and word representations that capture distribution.
5. There are two main approaches to word representation: counting and learning (Word2Vec).
6. There are different neural networks as language models. Here, we present two classical examples: RNN and Seq2Seq.

References

- Christopher Olah'S Blog for LSTM. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`
- Gensim Word2vec embeddings: `https://radimrehurek.com/gensim/models/word2vec.html`
- Jurafsky, D. and Martin, J.H., Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Ch 6. Vector Semantics and Embeddings. `https://web.stanford.edu/~jurafsky/slp3/6.pdf`.
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*
- Graham Neubig. Neural Machine Translation and Sequence-to-sequence Models: A Tutorial. `https://arxiv.org/pdf/1703.01619.pdf`.
- Stanford University CS224n: Natural Language Processing with Deep Learning. `https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/`